PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F

A2

(11) International Publication Number: **WO 98/25198**

(43) International Publication Date: 11 June 1998 (11.06.98)

(21) International Application Number: PCT/US97/22380

(22) International Filing Date: 6 December 1997 (06.12.97)

(30) Priority Data:
60/031,987 6 December 1996 (06.12.96) US

(71) Applicant *(for all designated States except US)*: STREAMIX CORPORATION [US/US]; 550 15th Street, San Francisco, CA 94103 (US).

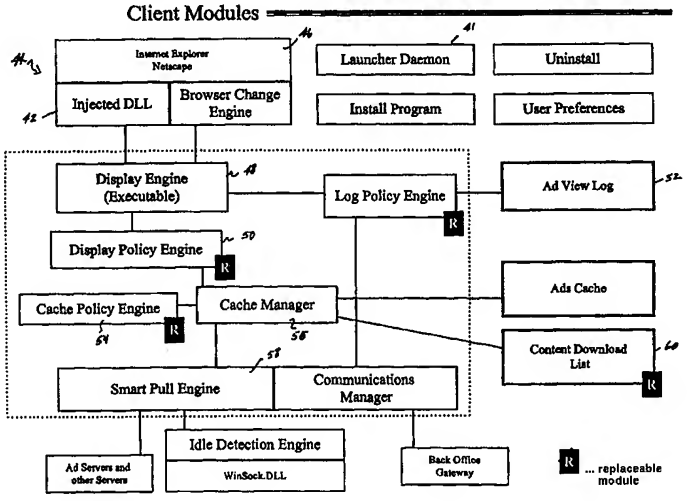
(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: SHIN, Kap, Jung [US/US]; 1388 Gough Street #705, San Francisco, CA 94109 (US). KIM, William, E. [US/US]; Suite 427, 200 Roy Street, Seattle, WA 98109 (US). BABBITT, Victor, Leroy [US/US]; 2171 Dailey Street, Superior, CO 80027 (US).

(74) Agent: JUDSON, David, H.; Hughes & Luce, L.L.P., Suite 2800, 1717 Main Street, Dallas, TX 75201 (US).

(81) Designated States: AU, BR, CA, IL, JP, KP, KR, MX, US, European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
Without international search report and to be republished upon receipt of that report.

(54) Title: INTERSTITIAL CONTENT DISPLAY USING EVENT-CAPTURE CODE RUNNING IN WEB BROWSER ADDRESS SPACE



(57) Abstract

An interstitial content display system and method for a Web browser wherein given executable code (42) shares the Web browser's address space (44) and is used to capture Web browser events. The Web browser events, in turn, are used to drive a display engine (48) that generates a viewer window. The viewer window preferably includes appropriate user interface controls, and it is re–positionable and re–sizable both manually and automatically. In operation, the executable code is responsive to a given browser event, such as activation of a URL in a source Web page, for controlling the display engine to overlay the viewer window on the browser display window and to display given content therein during the interstitial delay period while the browser waits for return of the target Web page. Upon a given occurrence, such as receipt of the target Web page (or some portion thereof), the user window may be selectively hidden or re–sized and re–positioned (into a so–called "mini–window") to allow the interstitial content display to complete without interfering with the user's viewing of the target Web page.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

INTERSTITIAL CONTENT DISPLAY USING EVENT-CAPTURE CODE RUNNING IN WEB BROWSER ADDRESS SPACE

This application is based on and claims priority from U.S. Provisional Application Serial No. 60/031,987, filed December 6, 1997.

BACKGROUND OF THE INVENTION

Technical Field

The present invention relates generally to communications within a client-server computer network and, in particular, to a method of displaying content during Web page linking that utilizes a viewer-based user interface.

Description of the Related Art

The World Wide Web is the Internet's multimedia information retrieval system. In the Web environment, client machines effect transactions to Web servers using the Hypertext Transfer Protocol (HTTP), which is a known application protocol providing users access to files (e.g., text, graphics, images, sound, video, etc.) using a standard page description language known as Hypertext Markup Language (HTML).

Web browsing can be a frustrating experience for a user because of long delays between Web page downloads. Each time the user clicks on a hyperlink or enters a Uniform Resource Locator (URL) address, he or she has to wait for the data comprising the target Web page to be downloaded from a particular Web server in the network.

During this time, the user is either staring at content in the source page (i.e., the page on which the hyperlink was launched) or a blank screen. This delay, termed the "*interstitial*" delay, is becoming worse as the level of Web page complexity increases.

The Internet is a packet-based network of host computers, routers and electronic interconnections. Computers and routers can have different performance characteristics, and interconnections can have different bandwidth capacities. For these reasons, latency or delay is inherent in Internet communications.

This latency can be separated into "response" latency and "download" latency. Response latency is the duration from the time the Web browser sends a request to a Web site to the time the browser receives the first return packet from the Web site. Download latency is the duration from the time the first packet is received to the time the last packet of the requested file is received.

The overall latency that a given user experiences in downloading data from a given Web server is generally determined by several components: bandwidth capacity and traffic load of each interconnection between the user and the Web server, the speed and load of each router between the user and the Web server, and the speed and load of the host computer at the Web server. Obviously, latency can be a highly variable figure at any given time of the day or browsing session. The use of faster network

connections, moreover, does not necessarily reduce the interstitial delay. Even as connection speed is enhanced, such enhancements are often offset by the larger page content volume provided by dynamic HTML and other page scripting techniques. Moreover, connection speed only reduces response latency. It has been shown empirically that it is generally impossible to know *a priori* how long the *latency* will be with respect to a particular connection at any given time during a particular browsing session. In a typical Web browsing session, however, it has been shown that just the response latency for HTML documents ranges from **2** to **6** seconds. Download latency is obviously determined by the size of the file being downloaded.

As a consequence, each time the user requests data from a Web site, the user waits and focuses at the screen for at least some period of time, often waiting impatiently and passively for the Web browser to complete its activity.

It is known in the art to exploit this user "wait focus time" during the latency periods to display information to the user right inside the Web browser client window display area. The basic technique is illustrated in U.S. Patent No. **5,572,643**, for example, wherein an "information object" (sounds, videos, text, messages, animations, and the like) are displayed in and during the interstitial "time-space". This technique optimally takes advantage of the fact that there is wait

time and that the user's eye focus is on the client window area.

Although the basic approach of outputting content in the interstitial space is thus known in the art, there remains a need to provide enhanced control over display of such content, as well as additional methods and techniques for enhancing the user's ability to direct the nature and types of content being displayed during the interstitial delay period.

BRIEF SUMMARY OF THE INVENTION

It is a primary object of this invention to provide an interstitial content display system and method for a Web browser wherein an given executable code shares the Web browser's address space and is used to capture Web browser events. The Web browser events, in turn, are used to drive a display engine that generates a viewer window. The viewer window preferably includes appropriate user interface controls, and it is re-positionable and re-sizable both manually and automatically. In operation, the executable code is responsive to a given browser event, such as activation of a URL in a source Web page, for controlling the display engine to overlay the viewer window on the browser display window and to display given content therein during the interstitial delay period while the browser waits for return of the target Web page. Upon a given occurrence, such as receipt of the target Web page (or some portion thereof), the user window may be selectively hidden or resized and re-positioned (into a so-called "mini-window") to allow the interstitial content display to complete without interfering with the user's viewing of the target Web page.

In accordance with a preferred embodiment of the invention, a method of displaying interstitial content is operative in a client computer connectable to a plurality of Web servers via a computer network, the client computer supporting a Web browser for controlling display

of Web pages within a display window. Upon launch of the Web browser, given event monitoring code (e.g., a DLL) is injected into the address space in which the Web browser is running. If a given browser event is then captured by the event monitoring code, the code passes control information to a display engine that overlays a viewer window on a given portion of the browser display window. One or more content pieces are then displayed within the viewer window during an interstitial delay period.

Thus, for example, the event monitoring code typically captures a user's activation of a link to a URL identifying a page on a target Web server. The event monitoring code passes the event to the display engine to activate the viewer window to display one or more interstitial content pieces in the viewer window while the browser "waits" for a given display termination event to occur. The given termination event may be receipt by the browser of a first packet of data comprising the target Web page (identified by the URL) or, more preferably, some given browser action with respect to that Web page. Thus, for example, the given browser action may be the drawing of a first portion of text in the target Web page, or a first image (e.g., .gif or .jpeg file) or the like. Upon occurrence of this termination event, the viewer window may disappear or shrink down and away from the main browser display window so as not to obscure the user's view of the target Web page content that has now been received by the browser.

The inventive method does not interfere with or

otherwise delay the normal operation of the browser. All of the actions are carried out in a transparent manner to the Web browser code. The user, however, is provided with an enhanced browser experience due to the display of content in the otherwise useless interstitial time-space.

It is thus an object of this invention to provide a user with significant control over content display within and during the interstitial time-space as Web page linking activity is carried out by a Web browser.

It is a more specific object of the invention to provide a user interface viewer for use to facilitate display of user-selectable content pieces during Web inter-page connections during a browsing session.

A still further more general object of the invention is to enhance a user's Web browsing experience by enabling the user to determine what types of imagery (e.g., advertising, editorial and news content, "cool" content, images, vector animations, messages, graphics, videos, interactive question/response games and the like) that will be displayed during the interstitial delay periods.

Thus, another basic object of the invention is to transform the user's wait-browse-wait activity sequence to watch-browse-watch sequence, in effect replacing the frustrating delays with entertainment and information.

The foregoing has outlined some of the more pertinent objects and features of the present invention. These objects should be construed to be merely illustrative of some of the more prominent features and

applications of the invention. Many other beneficial results can be attained by applying the disclosed invention in a different manner or modifying the invention as will be described. Accordingly, other objects and a fuller understanding of the invention may be had by referring to the following Detailed Description of the Preferred Embodiment.

BRIEF DESCRIPTION OF THE DRAWINGS

For a more complete understanding of the present invention and the advantages thereof, reference should be made to the following Detailed Description taken in connection with the accompanying drawings in which:

Figure 1 illustrates the conventional browse-wait-browse experience with a known Web browser implementation;

Figure 2 illustrates the "browse-watch-browse" experience that is achieved using the present invention;

Figure 3 is a block diagram of the interstitial content display system and method of the present invention;

Figure 4 is a flowchart of a preferred method of interstitial content display according to the present invention;

Figure 5 is an illustration of the "mini-window" operation of the present invention;

Figure 6 is a more detailed block diagram of the interstitial client application and its associated

modules;

Figures 7A and **7B** are representative Preferences dialog screens that may be used to set or select individual viewer window display characteristics and display object types, respectively; and

Figure 8 illustrates a modified display method of the invention wherein an interstitial advertisement is integrated with a known "banner" advertisement on a Web page.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

Figure 1 illustrates the current Web browsing experience due to the interstitial delay period. In this example, activation of a link in the page being viewed (i.e., the source Web page) causes the Web browser to pull a new page. The user, however, must "wait and stare" between Web page accesses. On the contrary, **Figure 2** illustrates the display of interstitial content (e.g., advertisements, cool images, interactive games, video clips, and the like) so that the user experience is now "browse-watch-browse".

The interstitial content display system of the present invention is illustrated in the block diagram of **Figure 3**. It is preferably implemented in an existing Internet-based client-server system. Familiarity with the basic operation of such a known system is thus presumed in the following discussion.

The invention comprises a client application portion

10 that is downloaded to or otherwise supported on the user's "client" machine 12 in the network. The client application displays one or more content "pieces" during the delay between Web page linking. This display fills the user's browser window, and the content pieces may be quite varied such as display of animations, movies, static information and interactive content. The imagery that is displayed is preferably pre-cached on the user's computer by the client program which, as will be described below, "smart-pulls" the content pieces onto the client's hard disk over time from an ad content server 14. This content download does not slow the user's use of the browser, however, as will be seen.

The particular list of content pieces to display, and the policy about which particular piece that is displayed during a page switch, may be dynamically changed by a back office 16, which communicates with each active client program. The back office 16 provides centralized control of client applications running on a large number of client machines throughout the computer network. It preferably controls what content pieces a specific client downloads, as well as providing automatic revisions of how the application decides what content piece to display and how the client's cache is managed. The back office may also gather statistics on how often certain content pieces were viewed. Thus, preferably, the display of each content piece is logged at the client's computer, and the log information, along with

user demographics and preferences, is uploaded to the back office on a periodic basis. **Figure 3** illustrates the flow of control and data between the back office, the content ad content server, advertisers and content providers (who supply content pieces) and the client application.

The interstitial client application may be distributed as an ActiveX control or a Netscape plug-in, or through any other convenient means. Alternatively, the functionality described below may be built into the browser itself, in whole or in part. Once installed, the client program runs in the background whenever the Web browser is in operation as will be described. The user can set various options such as the amount of disk space used for downloaded content, different navigational imagery themes, and the like.

Figure 4 is a flowchart illustrating the basic "browse-watch-browse" experience provided by the client application **10**. When the user is using the Web browser application, his or her eyes are focused on the client window area. When the user specifies a new URL address or clicks on a hyperlink on a Web page at step **20**, the Web browser first attempts to resolve the domain name in the associated URL into an IP address by contacting a known DNS server. The DNS attempts to resolve the URL and returns the IP address to the Web browser. The Web browser then sends the data request to that IP address using the HTTP protocol. At step **22**, the interstitial

client application traps the Web browser's hyperlink event and opens a separate window (the "viewer window") to cover some defined portion of the Web browser's main window. Next, at step **24**, the interstitial application loads an image from the local image cache and, at step **26**, causes the image to be displayed in the viewer window. The types of image data may include, without limitation, static bitmaps, animations, multimedia sound and video, and Java applets. The viewer window may include buttons, edit boxes, or other user-interface controls for allowing interaction with the user.

Step **28** represents the Web browser's receipt of a packet from the Web server identified by the URL. Next, upon a given occurrence (e.g., when enough of the target Web page has been received to be useful to the user) as indicated at step **30** (or perhaps after the response latency has elapsed), the routine preferably scales down and moves the viewer window off the Web browser window. This operation is step **32** in **Figure 4**, and it is illustrated in **Figure 5** by way of example only. Optionally, the application can be programmed to scale down the viewer window after the complete latency (response latency + download latency) has elapsed, or under some other given criteria.

As seen in **Figure 5**, this scale down or "mini-window" feature enables the content piece to continue to play to its completion in the smaller viewer window off

the Web browser. The Web browser then renders the new Web page in its client window area at indicated by step **34** in **Figure 4**. As also indicated in the flowchart, during the interstitial content display, the user may click on the image shown as indicated at step **36**. When the user clicks on the image, the application opens a new Web browser window and loads the URL associated with the image.

In a separate execution thread, the inventive system runs the "smart-pull" thread. This thread monitors network data traffic in the user's computer. If data traffic is low, as indicated by an outcome of the test at step **38**, the thread continues at step **40** to download the next image specified in a content download list from specified FTP servers on the Internet. As indicated by step **42**, the downloaded images are stored in a local cache directory in the user computer's hard disk.

A more detailed block diagram of the various functional modules and components that comprise the client application **10** is illustrated in **Figure 6**. As noted above, the client application of the present invention provides a user with images during the interstitial time of a browser. In order to provide this functionality, the application must be able to detect what the browser is doing at any time. To achieve this, there are three (3) primary modules that comprise the application: the launcher daemon, the display engine and

injected DLL, as well as a number of ancillary of support modules for determining which display content management.

The launcher daemon **41** is a process that preferably runs at all times, even when the 3rd party browser is not running. For the most part then, it is an invisible application. Its main purpose it to detect when a 3rd party browser is launched. Upon such detection, the launcher places or "injects" the injected DLL **42** in the address space **44** that the operating system has established for the 3rd party browser **46**. This is preferably accomplished as follows. The launcher daemon is placed in the "Run" key of the system registry, so that the Windows (or other) operating system will load it automatically each time the operating system is started. Every few seconds, the launcher enumerates through the entire list of windows in the system to see if any of the windows belong to an appropriate 3rd party browser **46**. If so, it uses the Windows OS message hooking mechanism to place the injected DLL **42** in the same address space of this 3rd party browser (typically at the first available location). This injection technique is standard and is documented by Microsoft. The launcher is also responsible for launching the display engine **48** each time a new browser window appears.

The display engine **48** is responsible for displaying images and other interstitial content to the user. As

will be seen, the engine **48** depends heavily on the Injected DLL **42** to learn the state and location of the browser at any given time. Thus, the injected DLL functions to "capture" certain Web browser events. The display engine **48** contains two primary threads: a main window thread for processing normal windows events, and a tracking thread that handles communication with the injected DLL. This communication technique is described below.

The display engine **48** generates the viewer window and makes itself topmost while displaying images so that it will appear over the browser. However, the display engine viewer window is not activated so that the use can still interact with the browser. If the browser becomes inactive, then the display engine is not longer top-most window.

As noted above, the Injected DLL **22** is located in the address space **44** of each 3rd party browser **46** running on the system. Once the DLL is running inside the 3rd party browser, it is able to detect windows messages and to reroute function calls. The DLL becomes a passive component of the 3rd party browser, however, the injected DLL does not affect the performance or behavior of the 3rd party browser. The injected DLL's primary purpose is to report to the rest of the application what is happening inside the browser. Preferably, the injected DLL is not placed into the address space of the 3rd party browser

until well after the browser has completed loading its own components.

The display policy engine **50** is also preferably a .dll that exports a function that takes a URL variable and returns an identifier of a cached ad (a path or other identifier that the display engine **48** can use to find a particular content piece. The display policy engine **50** thus determines what content piece to play during a certain interstitial delay. Many different policy engines may be used. Thus, for example, one policy simply returns a random cached content piece. Alternatively, the display policy engine may make a display decision based on any or all of the following: the URL, the time/date, statistics found in a content download list or statistics found in an ad view log **52**. These data are either found directly or by calling exported .exe or "core" dll functions. Yet another alternative may use user demographic data to determine a content piece.

The cache policy engine **54** decides which content piece to discard in the event the total data size of all content piece exceeded maximum (e.g., user selectable) cache sizes. Many different cache policies may be used, and a particular cache policy may be closely tied to a particular display policy. A cache manager **56** operates in conjunction with the cache policy engine **54** for this purpose.

The smart-pull engine **58** downloads content pieces as defined in the content download list **60** to the user without severely affecting the user's Web browsing. This operation has been previously described in conjunction with the flowchart of **Figure 4**. Preferably, the engine starts the transfer of content pieces during modem idle time, and it stops data transfer quickly should the user begin to use the Web browser. Content piece transmission then restarts where a particular prior transmission was interrupted. Because the content pieces may reside on servers under third party control, the smart-pull engine preferably uses a commonly available protocol. The smart-pull engine may also be used to download new display policy engines and cache policy engines.

The content download list **60** is the list of URLs of content pieces to download. In addition, each URL may include a ClassID identifier (e.g., GUID or similar) and the playing length in time of each content piece (which, of course, may differ). The entire content download list preferably has a version number, which can be compared to the latest revision number obtained from the back office to allow the application to know when to download a new content download list. By storing content attributes in the content download list, the system avoids having to attach fields to each content piece individually. The ad view log **52** is the data structure where the display engine logs ad viewing statistics. This may simply be a list of ads and a number field that is incremented to

note number of viewings of a certain ad, along with log start and stop dates. This allows very small ad view logs that are easy to upload to the back office. In one example, the ad view log is simply a log of all viewing events, with the ad viewed, the time and date, and what URL the user clicked. The entire log should have an effective start date as well.

The back office preferably is separate from the content server that transmits the display content. The URL for the ads is stored in the content download lists, which are smart-pulled from the back office by the client application, which then smart-pulls the display content pieces themselves. Content pieces may be stored at a given server in the network.

Thus, according to the present invention, the interstitial display system and method uses a unique process for inserting an external program (the DLL) into a host application process (i.e., the Web browser) and executing such program during certain pre-defined events (e.g., a user's activation of a link, re-sizing of the browser window, receipt of first packet return in response to the link activation, display of text from the target Web page, etc.). As a by-product, information (such as advertisements, cool content, and the like) is displayed directly in front of the computer user's line of sight. By use of the viewer window, there is improved placement of information from a background application (the Web browser) directly in-front of the user's eye focus during times when the user is waiting for the

browser application to do some activity.

The following describes the injection and event capturing mechanism of the injected DLL and display engine in more detail. In a preferred embodiment, there are several "methods" that the client application can observe with respect to a 3rd party application (such as a Web browser). These include: calls to Windows functions that take "HWND" as a parameter; the method that adds the interstitial application **10** to the message hook chain for the 3rd party application, rerouting 3rd party window functions to the application, rerouting 3rd party DLL function calls to the application, and rerouting V-table function calls to the application. The application **10** preferably uses all of these methods. In particular, method **1** can be used by the Display engine; methods **2-4** require the application to inject the DLL in the 3rd party browser as previously illustrated. Each of these methods is now described.

Method **1**: The display engine information from the browser by calling normal window API functions that take an HWND as a parameter. For example, the display engine could find out if the browser was still running by calling IsWindow (browserHwnd).

Method **2**: Message hooks are a standard windows method for a set of code to observe the messages that are sent to any windows in the system. A message hook receives all messages targeted to all windows in a

process. A beneficial side effect of message hooks is that the observing code is automatically injected into the process space of the observed window. For example, the injected DLL uses this method to detect the creation of new windows within a browser.

Method 3: Each window contains a pointer to a window function. The interstitial application in some cases replaces this pointer to its own function, thus rerouting the call to the application. When the application has gathered the information that it needs, it then sends the message on to the original function call. This method is how the application determines that the browser's status bar has changed.

Method 4: Every module that is statically linked to another module contains an import table. The application can locate this table, search for the desired function, and the replace that entry in the table with a pointer to a corresponding interstitial application function. The application version of the imported function eventually calls the original imported function. For example, this is how the application determines that the 3rd party browser has started writing text to its own display.

Method 5: Some programs, especially those written in C++, contain tables of function pointers to internal functions. The application is able to obtain the location of these pointers through standard function calls and then can reroute the functions to the application versions. Afterwards, the application calls

the original function. For example, this is how the application can tell when a new interstitial has begun.

The Injected DLL preferably communicates with the display engine in the following manner. Once the application determines that browser's state has changed, it needs to report that change to the display engine. The reporting mechanism typically requires a memory mapped file that can be seen from many different address spaces. The injected DLL writes to this memory mapped file to report any relevant changes in the browser, such a mouse click. Then the injected DLL sets a mutex that triggers the display engine to read this memory mapped file.

The following steps describe this process in more detail.

1. The 3rd party Browser reaches a new state, either through user interaction or internet communication

2. The application injected DLL detects this new state.

3. The application injected DLL acquires ownership of the memory mapped file through a mutex.

4. The application injected DLL updates the memory mapped file with the new browser state information.

5. The application injected DLL notifies the display engine that the memory mapped file has been updated using an system event object.

6. The application injected DLL releases ownership

of the memory mapped file.

7. The display engine receives the event notification.

8. The display engine gains ownership of the memory mapped file.

9. The display engine reads the memory mapped file and acts on it.

10. The display engine releases ownership of the memory mapped file. This completes the processing.

Thus, the injected DLL out of the way of the browser, and only display images when the browser is causing the user to wait. In order to do this, the injected DLL must account for many events and conditions of the browser. The following is a list of some of the conditions of the browser that must be tracked: browser location, browser size, what HWNDs the browser has created, what threads the browser has created, and the status of the browser's page download timer.

The following is a list of the browser events that may be tracked by the Injected DLL: a new window is created within the browser, the browser's status bar has updated, the browser's download timer has started, the browser's download time has finished, the browser is moving to a new URL, the browser has erased a page, the browser has drawn the wallpaper of a new page, the browser has drawn a graphic on a page, the browser has drawn text on a new page, and, the browser has become active or inactive. These (and perhaps other browser

events) then trigger some action with respect to the viewer window as will be described.

The following are representative events that are reported to the display engine: the browser is unloading, the browser has changed size or location, the browser has started downloading a new URL, the browser has started drawing a new URL, the browser has finished drawing a new URL, and, the browser has become active or inactive.

In the present invention, when the injected DLL captures a given Web browser event, some given action may occur with respect to a "viewer window" used to display content pieces during the interstitial delay period. The viewer window may go away when the first packet of information (from the target Web page) has come back. A better approach, however, is to look to see when "enough" of a page has been downloaded to be viewable. As noted above, the application knows when the user has clicked on a new URL. There are many ways that a user can start a new Web page, including typing in a new URL, clicking on a hypertext link, choosing a menu item such as "favorites," or even using automated slide show html commands. In all cases, the application distinguishes between a true new URL event and a red-herring event (such as when a Java applet downloads and displays new banner ads while it is running). Red-herring events are thus ignored. Also, the application looks to see if a page is cached. If so, it preferably ignores the URL event.

Once a new URL event is detected, the display engine

starts displaying an image. The injected DLL then continues to look for the browser to start drawing the new page. Once the application determines that a new page has started drawing, it may look for either text to be drawn or a bitmap to be drawn before taking some action with respect to the viewer window (such as closing it or entering the "mini-window" operation). In the case of Netscape browsers, wallpaper drawing can be ignored. In one exemplary embodiment, the viewer window stays in place with the content piece continued to be played even if the browser window is erased and wallpaper is drawn.

The injected DLL may then decide that enough of a target Web page has been loaded as soon as any text is drawn to the screen. Because some pages do not include any text at all, the application may alternatively look to see if any images (besides wallpaper) are being drawn to the screen. If so, it finishes playing the current image while the image is being downloaded. If the application is still playing an image in injected mode, and a page completes downloading, then the application preferably leaves injected mode. Of course, the above examples are merely exemplary. The adjustment of the viewer window (e.g., its disappearance, repositioning or resizing, as the case may be) may depend on any particular Web browser event relative to the target Web page download.

As seen in **Figure 7A**, the application preferably includes a preferences dialog box that allows the user to

control the behavior of the viewer window. **Figure 7B** illustrates a dialog box that may be used to select various "themes and ratings" for the content pieces. In addition, the viewer window preferably includes "back" and "forward" buttons so that the user may view the content pieces directly. In a preferred embodiment, an "icon" representing the application may be displayed on the user's desktop. When this icon is activated, the display engine starts the viewer window in an "off-browser" mode so that the user may view the content pieces even if the client machine is offline.

The following viewer window display options are preferably available to the user, although one of ordinary skill will appreciate that other control functions may be implemented. Each option is described below.

As noted above, "injected mode" is the state during which the display engine generates a viewer window that covers the active display area of the Web browser on the graphical user interface. Thus, typically "injected mode" is entered when the user activates a link in a source Web page (or otherwise initiates some given action to pull a target Web page). Unless the user has set a different preference, the viewer "default" operation during this mode preferably is to observe the location and size of the browser window and then match it. If at any time the browser is moved or resized, the viewer engine gets a message, which preferably causes the viewer

to then move or resize itself to continue to cover the Web browser's client display window.

For injected mode, the user may (using a preferences dialog box) choose to cover the entire client window, or to cover some given percentage (e.g., **90%**) of the client window. The preferences dialog box was seen in **Figure 7A**. The benefit of **90%** mode is that the user can see enough of the browser client screen to assure that the requested target page has yet to be downloaded. Preferably, when in **90%** mode, the viewer is centered in the middle of the browser's display window, and thus some small percentage of the browser window is still visible on each of the four sides.

During injected mode, at least one display object (e.g., an image, an animation, an interactive form, a video clip, a sound bite, or the like) is output by the viewer window directly in front of the user's eye focus to fill-in the interstitial period. A particular object may have a given time duration. Thus, when a first display object is finished playing (which, of course, depends on its given time duration and the length of the interstitial delay period), the display engine will load and play a new object, and so on, as long as the viewer remains in injected mode. Thus, if there is an extended interstitial delay (e.g., if the user has initiated a search function that requires extensive server processing to return a result), the viewer display engine outputs multiple information objects.

When enough of a page has been downloaded, the viewer exits or leaves injected mode. Using the preferences dialog box shown in **Figure 7A**, the user preferably also selects what the viewer does upon exiting injected mode. One choice is for the viewer to disappear entirely. Alternatively, the user may choose to let the current image finish playing or the user may choose to have the image abruptly stopped, even before it has finished playing. In the latter case, the user will not see the entire image. If the user (based on a given preference setting) lets the current image finish playing, then he or she is giving the display engine viewer permission to play an image past the interstitial time.

While the browser is the active window, the injected mode viewer is the topmost window so that it appears above the browser. When the browser becomes inactive, the viewer window is no longer topmost, but it preferably still appears above the browser. A browser may become inactive if the user clicks on another application.

When enough of a page has been downloaded, the user can choose for the viewer display engine to switch to "mini-window" mode using the dialog preferences box. As previously illustrated, in this mode, the viewer window is moved out of the user's direct eye focus but still finishes playing the current image. In particular, the mini-window is reduced to a smaller size so that the user can see the browser. The location and the size of the

mini-window can be changed using standard windows-based mouse click and drag operations. The mini-window preferably "remembers" this location and appears wherever the user last left it.

The transition from injected mode to mini-window mode preferably uses the standard windows "zoom effect", which draws an animated rectangle that moves from the old location to the new location. In particular, the injected mode window is first hidden, the zoom effect is started, and then the viewer window is shown at its new mini-window size. This operation was illustrated in **Figure 5**. As illustrated there, the zoom effect preferably starts with the injected mode size and location and ends up with the mini-window size and location.

When the mini-window finishes playing the current image, the user also preferably has the option of leaving the mini-window visible or hiding it. If the mini-window is left visible, then the last frame of the image preferably is displayed indefinitely. While in mini-window mode, the user also preferably has the option to look at previous images. Thus, for example, if the user press a `back' or `forward' button associated with the viewer window, then an image that has already been shown will be played again. When the image is finished playing, the last frame preferably is displayed indefinitely.

The user may enter off browser mode by selecting a

menu choice or pressing a button. When in off-browser mode, the viewer window preferably fills up ½ of the user's screen. The viewer window, however, preferably does not track the browser and the viewer does not change behavior if the browser becomes inactive. In off-browser mode, the user can look at previous images and new images by pressing the `back' and `forward' buttons of the viewer. This mode also gives the user access to other functions, such as printing an image or saving it to disk.

The viewer is preferably implemented as a state machine although this is not a requirement. The following is a brief description of each state and each event that can change the state.

1. s_hiddenStopped - the viewer is hidden and is not loading an image;

2. s_alreadyHiddenLoading - the viewer is hidden, but is loading an image from disk;

3. s_maxFrontVisiblePlaying - the viewer is in injected mode, the browser is active, and an image is playing;

4. s_maxFrontVisibleLoading - the viewer is in injected mode, the browser is active, and an image is loading. The last frame of the previous image is still displayed.

5. s_minFrontVisiblePlaying, - the viewer is in mini-window mode, the browser is active, and an image is playing;

6. s_minFrontVisibleStopped - the viewer is in mini-window mode, the browser is active, and an image has finished playing;

7. s_minFrontVisibleLoading - the viewer is in mini-window mode, the browser is active, and a new image is being loaded from disk. The last frame of the previous image is still displayed;

8. s_minToMaxFrontHiddenLoading - the viewer was in mini-window mode but is now going to injected mode, the browser is active, and an image is being loaded from disk;

9. s_minToMaxBehindHiddenLoading - the viewer was in mini-window mode but is now going to injected mode, the browser is not active, and an image is being loaded from disk;

10. s_abortToMinFrontHiddenLoading - the viewer is hidden but is moving to mini-window mode, the browser is active, and an image is being loaded;

11. s_abortToMinBehindHiddenLoading - the viewer is hidden but is moving to mini-window mode, the browser is not active, and an image is being loaded;

12. s_maxBehindVisiblePlaying - the viewer is in injected mode, the browser is not active, and an image is playing;

13. s_maxBehindVisibleLoading - the viewer is in injected mode, the browser is not active, and an image is being loaded. The last frame of the previous image is

still displayed;

14. s_minBehindVisiblePlaying - the viewer is in mini-window mode, the browser is not active, and an image is playing;

15. s_minBehindVisibleLoading - the viewer is in mini-window mode, the browser is not active, and a new image is being loaded. The last frame of the previous image is still displayed;

16. s_minBehindVisibleStopped - the viewer is in mini-window mode, the browser is not active, and the last frame of the previous image is still displayed. A new image will not yet be loaded;

17. s_showHiddenLoading - the viewer is hidden, but it will go into off-browser mode. An image is being loaded;

18. s_showFrontVisibleLoading - the viewer is in off-browser mode and is loading a new image. The last frame of the previous image is still displayed;

19. s_showFrontVisiblePlaying - the viewer is in off-browser mode and is playing an image; and

20. s_showFrontVisibleStopped - the viewer is in off-browser mode and the last frame of the previous image is still displayed.

The following are the preferred events that can change the state of the viewer:

1. e_AdLoaded - an image has completed loading from disk and is ready to play;

2. e_AdFinished - an image has finished playing;

3. e_NewUrl - the user has selected a new URL;

4. e_DispUrl - a given amount or portion of the target web page has been downloaded;

5. e_ActivateToInterStitial - a browser just became active, and it is still in an interstitial;

6. e_ActivateToAllLoaded - a browser just became active that is not in an interstitial;

7. e_Inactivate - the browser just become inactive;

8. e_BrowserMin - the browser has been minimized;

9 e_HideButton - the user desires to hide the viewer;

10. e_ShowButton - the user desires to go into off-browser mode;

11. e_MaxButton - currently, the same as e_ShowButton;

12. e_MinButton, - the user desires to go into mini-window mode;

13. e_BackButton - the user desires to view the previous image;

14. e_ForwardButton - the user desires to view the next image.

As discussed above, the smart-pull engine is used to download content pieces. This is not a limitation of the invention, however. Another technique for downloading a

content piece such as an advertisement is by integrating the interstitial ad with a regular "banner" ad on a Web page. Web page banner ads are predefined areas on the Web page where images are displayed. The HTML language provides the <IMG> tag for displaying a graphical image on an HTML page. These images must be in a format that is supported by the user's Web browser. The most popular image formats are GIF and JPEG for static images and GIF89 for animations. The following is an example of an image defined using the IMG tag. It specifies a GIF image with a width of 400 pixels and height of 80 pixels.

```
<!-- Ad Start -->
<IMG SRC="image.gif" ALT="Information message"
WIDTH=400 HEIGHT=80>
<!-- Ad End -->
```

When the Web browser reaches this command in the page, it automatically downloads the image specified in the SRC= attribute and stores in the browser's local cache. The present invention may exploit this approach to integrate interstitial ads and Web page ads as follows:

1. The user goes to a Web site that has HTML pages with embedded images defined with the standard IMG tag, but with attributes WIDTH and HEIGHT of zero.

2. When the Web browser loads these HTML pages, it downloads the images into the local cache but nothing is displayed because WIDTH and HEIGHT are zero. These images are in effect hidden objects.

3. When the user clicks on a hypertext link (i.e.

in the next interstitial time-space), the inventive application loads the image just downloaded by the Web browser and displays it in the viewer window covering the Web browser.

4. After the response or download latency has elapsed, the Web browser displays the requested Web page. In that page, now an image area is defined (with the IMG tag with WIDTH and HEIGHT greater than zero) for displaying the same image downloaded in the previous Web page and displayed in the previous interstitial time-space. This page can also include a hidden image object to be displayed in the next interstitial time-space. From the user's perspective, the system would operate as illustrated in **Figure 8**.

As noted above, one of the preferred implementations of the invention is as a set of instructions (program code) in a code module resident in the random access memory of the computer. Certain components of the interstitial application (e.g. the DLL and the display engine) may be built into the Web browser. Until required by the computer, the set of instructions may be stored in another computer memory, for example, in a hard disk drive, or in a removable memory such as an optical disk (for eventual use in a CD ROM) or floppy disk (for eventual use in a floppy disk drive), or downloaded via the Internet or other computer network.

Having thus described our invention, what we claim as new and desire to secure by Letters Patent is set

forth in the following claims.

CLAIMS

1. A display method operative in a client computer connectable to a plurality of Web servers via a computer network, the client computer supporting a Web browser for controlling display of Web pages within a display window, comprising the steps of:

upon launch of the Web browser, injecting event monitoring code in an address space of the Web browser;

if a given Web browser event has been captured by the event monitoring code, overlaying a viewer window on a given portion of the Web browser display window; and

displaying, within the viewer window, a set of one or more display objects during an interstitial delay period.

2. The display method as described in Claim 1 wherein the given Web browser event is a user's activation of a link in a source Web page.

3. The display method as described in Claim 2 wherein the interstitial delay period is a period between the user's activation of the link and a given occurrence with respect to a target Web page identified by the link.

4. The display method as described in Claim 3 further including the step of adjusting a given characteristic of the viewer window upon the given occurrence.

5. The display method as described in Claim 4 wherein the given characteristic is visibility of the viewer window.

6. The display method as described in Claim 4 wherein the given characteristic is a size of the viewer window and its position relative to the Web browser display window.

7. A computer program product for use in a client computer connectable to a plurality of Web servers via a computer network, the client computer supporting a Web browser for controlling display of Web pages within a display window on a graphical user interface, the computer program comprising:

a first process running in a given address space established for use by the Web browser;

a second process controllable by the first process in response to capture by the first process of a first Web browser event for overlaying a viewer window on a given portion of the Web browser display window to facilitate display to the user, within the viewer window, of a set of one or more display objects during an interstitial delay period.

8. The computer program product as described in Claim 7 wherein the first process is an executable code

stub.

9. The computer program product as described in Claim 7 wherein the second process displays the one or more display objects.

10. The computer program product as described in Claim 7 wherein the first process is responsive to capture of a second Web browser event for adjusting a given characteristic of the viewer window.

11. The computer program product as described in Claim 7 further including a third process responsive for launching of the Web browser for injecting the first process into the Web browser address space.

12. The computer program product as described in Claim 11 wherein the third process is launched upon initialization of the client computer.

13. The computer program product as described in Claim 7 wherein the viewer window includes control means for enabling the user to adjust a sequence of display of the one or more display objects.

14. The computer program product as described in

Claim 7 further including a display policy engine for selecting a particular display object to display in the viewer window based on a given criteria.

15. The computer program product as described in Claim 7 further including a cache policy engine for selecting a particular display object to remove from a cache based on a given criteria.

Figure 1. Current Web Browsing

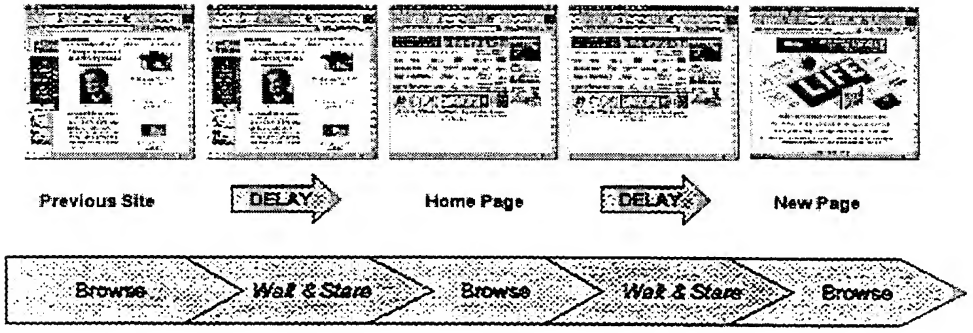
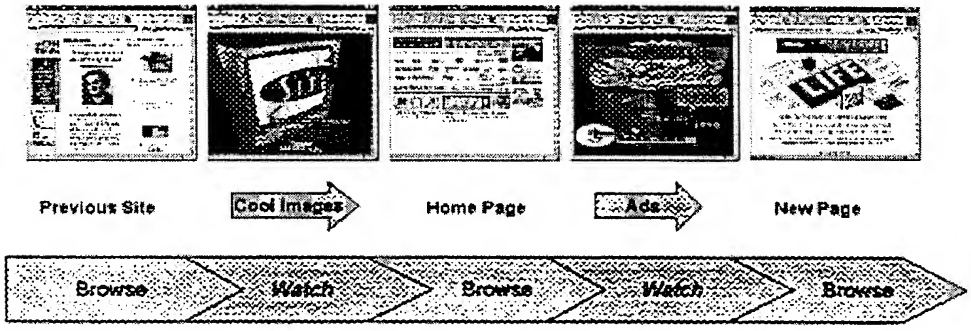


Figure 2. Web Browsing with HyperStitial System



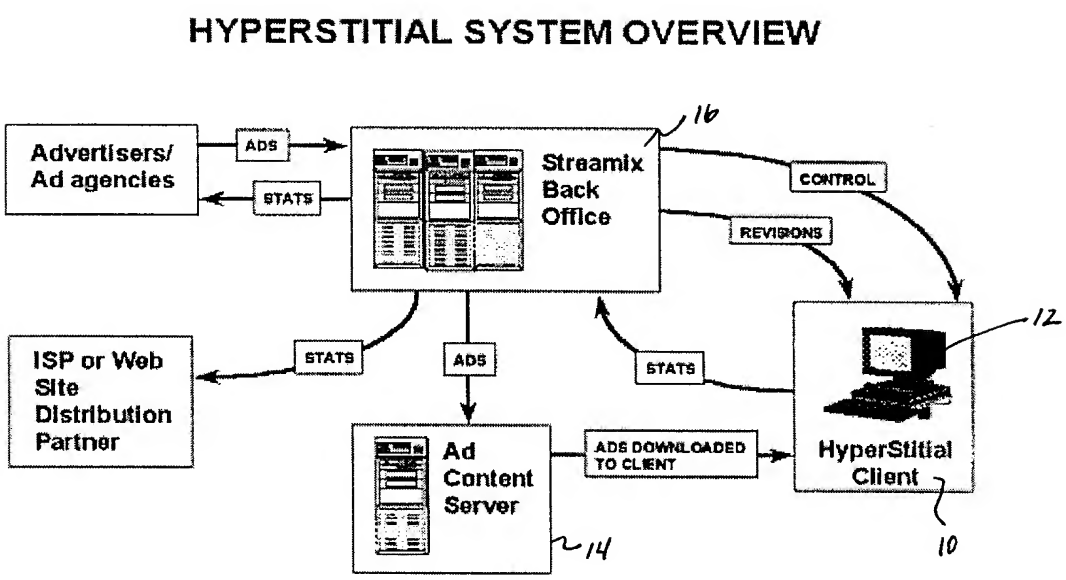

HYPERSTITIAL SYSTEM OVERVIEW
Advertisers/
Ad agencies
ADS
STATS
Streamix
Back
Office
16
CONTROL
REVISIONS
ISP or Web
Site
Distribution
Partner
STATS
ADS
STATS
12
Ad
Content
Server
ADS DOWNLOADED
TO CLIENT
HyperStitial
Client
14
10

Figure 3

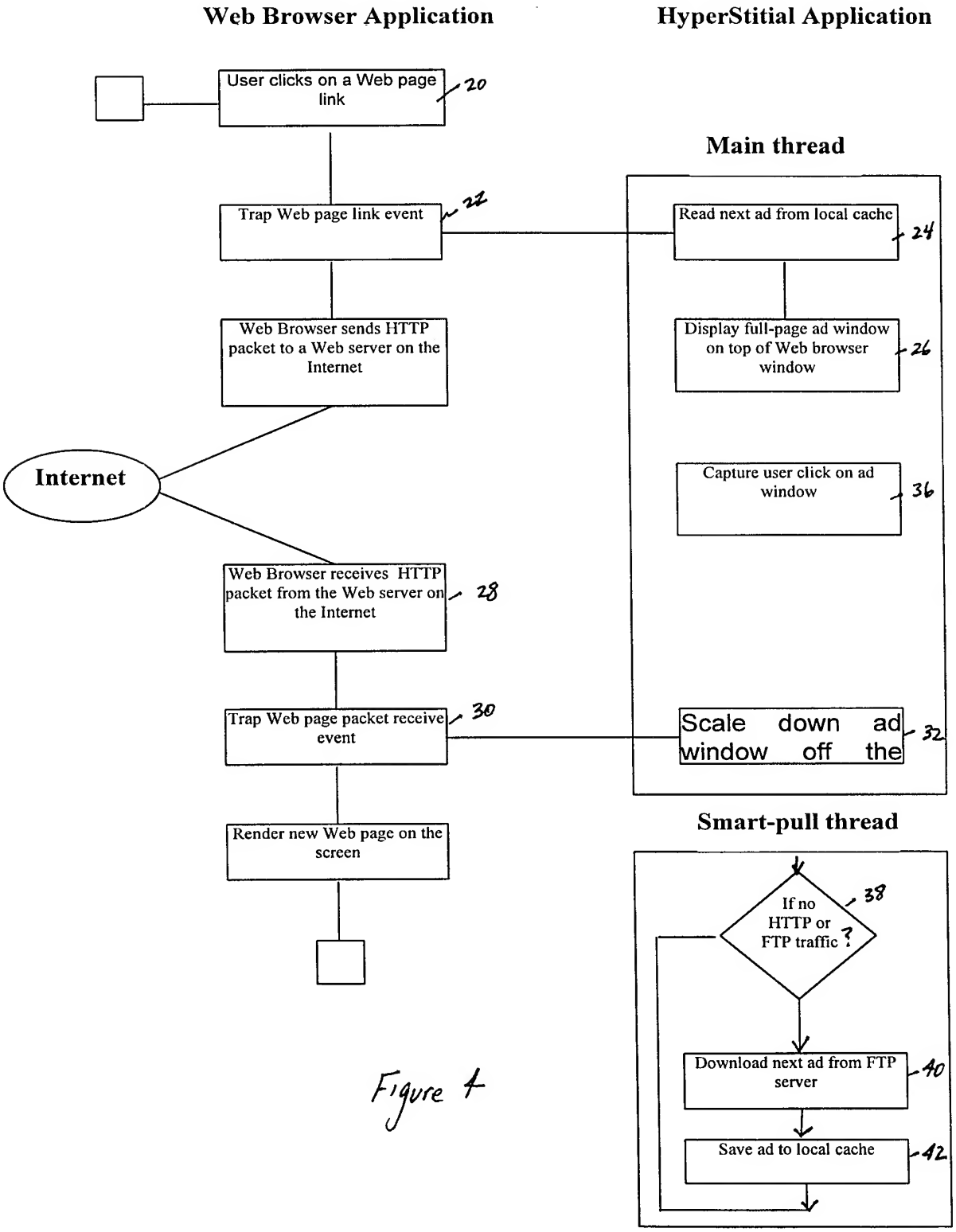

Web Browser Application
HyperStitial Application
User clicks on a Web page link
20
Main thread
Trap Web page link event
22
Read next ad from local cache
24
Web Browser sends HTTP packet to a Web server on the Internet
Display full-page ad window on top of Web browser window
26
Internet
Capture user click on ad window
36
Web Browser receives HTTP packet from the Web server on the Internet
28
Trap Web page packet receive event
30
Scale down ad window off the
32
Smart-pull thread
Render new Web page on the screen
If no HTTP or FTP traffic ?
38
Download next ad from FTP server
40
Save ad to local cache
42

Figure 4

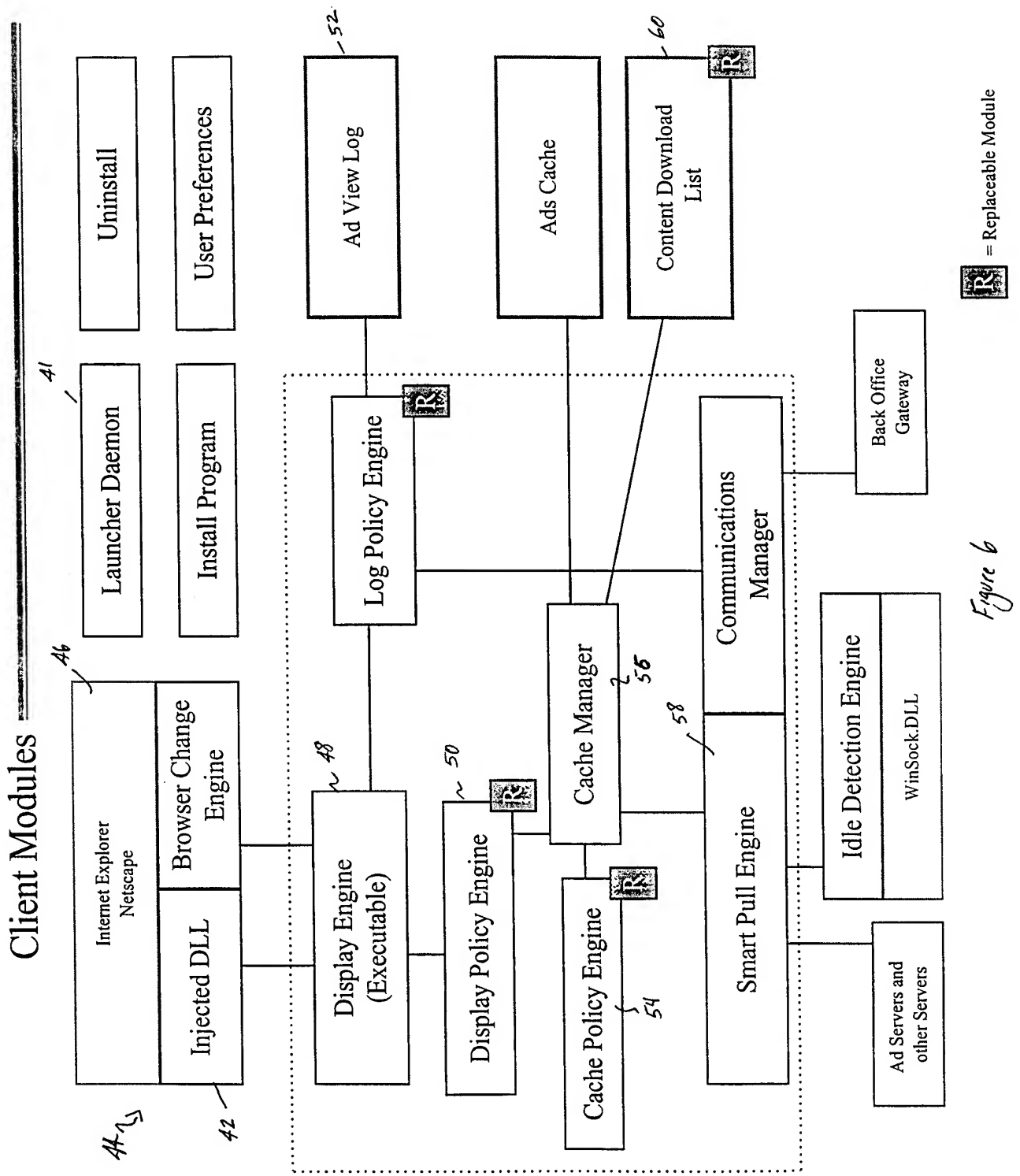

Client Modules
Internet Explorer Netscape
Injected DLL
Browser Change Engine
Uninstall
Launcher Daemon
Install Program
User Preferences
Display Engine (Executable)
Display Policy Engine
Cache Policy Engine
Log Policy Engine
Cache Manager
Smart Pull Engine
Communications Manager
Ad View Log
Ads Cache
Content Download List
Back Office Gateway
Idle Detection Engine
WinSock.DLL
Ad Servers and other Servers
= Replaceable Module
41
42
44
46
48
50
52
54
56
58
60

Figure 6

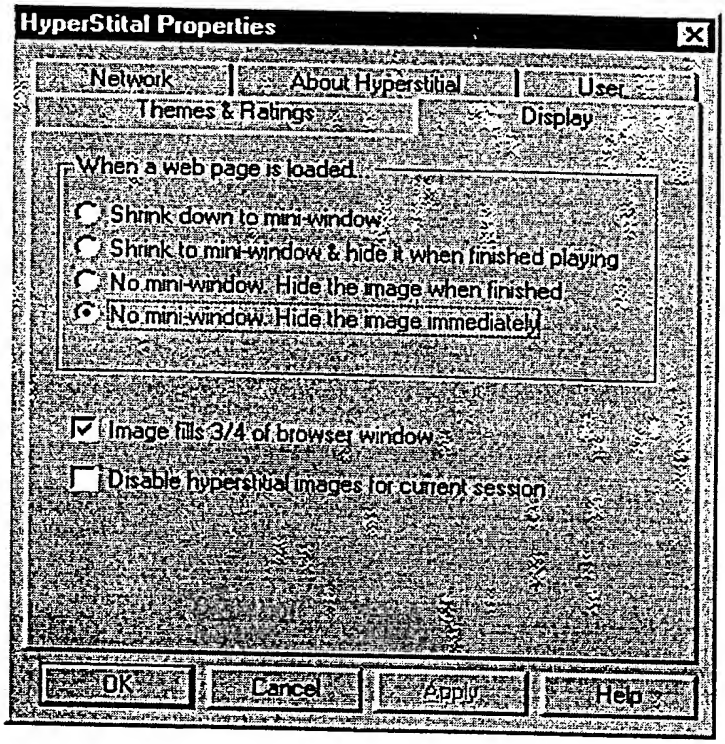


Figure 7A

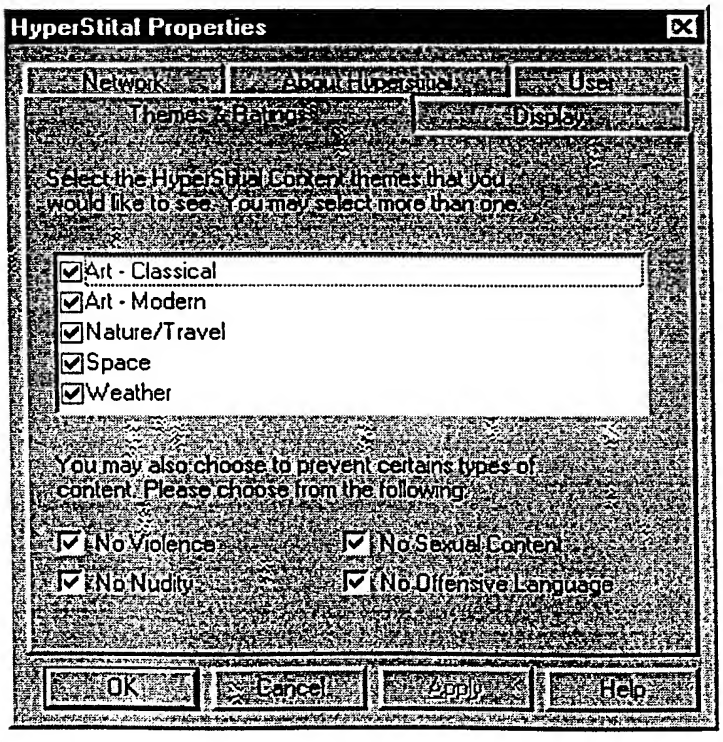


Figure 7B

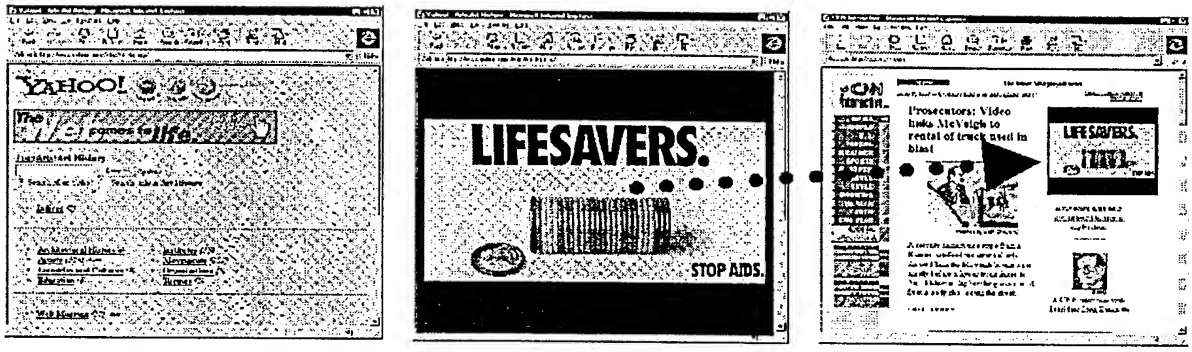


Figure 8

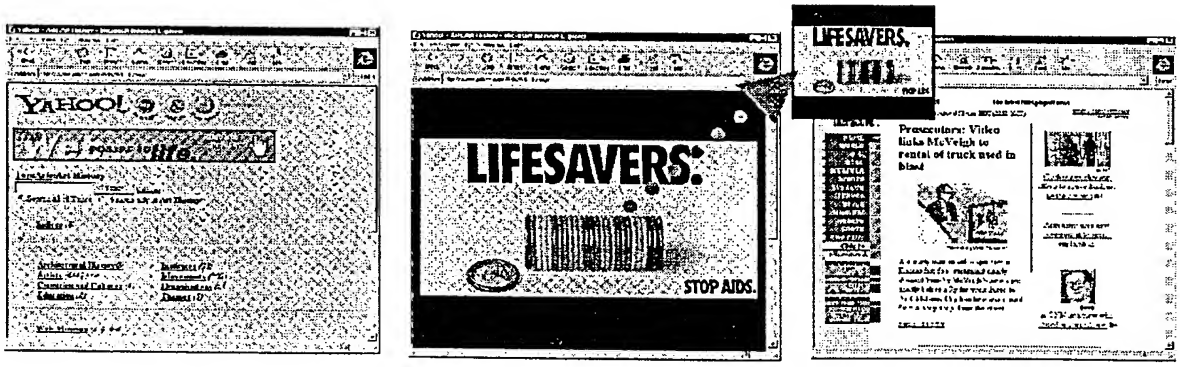


Figure 5